INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians
Universität
München
LMU

A Visual Language for Web Querying and Reasoning

Sacha Berger, François Bry, Sebastian Schaffert

Technical Report, Computer Science Institute, Munich, Germany
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-2003-6, June 2003

A Visual Language for Web Querying and Reasoning

Sacha Berger, François Bry, Sebastian Schaffert

Institute for Computer Science, University of Munich

Abstract. As XML is increasingly being used to represent information on the Web, query and reasoning languages for such data are needed. This article argues that in contrast to the *navigational* approach taken in particular by XPath and XQuery, a *positional* approach as used in the language Xcerpt is better suited for a straightforward visual representation. The constructs of the pattern- and rule-based query language Xcerpt are introduced and it is shown how the visual representation visXcerpt renders these constructs to form a visual query language for XML.

1 Introduction

Five years after its initial specification in 1998, XML [1] has become the de facto standard for data exchange. It is nowadays increasingly being used for representing semistructured databases, Web documents, and in particular meta information like ontological data (as in OWL [2]) or browsing contexts and user models [3]. There is hence a need for languages that are suitable for both querying and reasoning with semistructured data.

Many existing query languages, in particular the W3C proposals XPath and XQuery, are *navigational* in the sense that their variable binding paradigm requires the programmer to specify path navigations through the document (or data item). In contrast, some other languages – such as UnQL [4] and Xcerpt [5] – are pattern-based: their variable binding paradigm is that of mathematical logics, i.e. the programmer specifies patterns (or terms) including variables. This difference is discussed in Section 2.

In this article, it is argued that the pattern-based paradigm is particularly well-suited as a base for a visual query language for semistructured databases. The reason is that patterns are form-like two dimensional structures that conceptually are very close to two dimensional visual representations. Arguably, every visual or graphical language for XML and/or semistructured data (such as XML-GL [6], GraphLog [7], VXT [8], BBQ [9] and Xing [10]) as well as the veteran language QBE and improvements thereof (such as MS Access and similar products) might be seen as having an (in general implicit) pattern-based language as an (in general unconscious) foundation.

Interestingly, and maybe supporting the last above-mentioned claim, a visual language for a pattern-based textual query and transformation language can

be developed simply by specifying a visual *rendering* (in contrast to a complex transformation) of the textual programs very much like a CSS stylesheet specifies a layout for an HTML document.

Besides the pattern-based nature, another property of the Xcerpt language is of particular interest to visual querying: rule-based queries with a clear separation of condition and result allows for a rather natural visual representation, since an "if ... then ..." is easily conveyed even by novice users.

This article is organised as follows. Section 2 provides a discussion of the navigational and positional approaches for query languages. The basic elements of the declarative, rule- and pattern-based language Xcerpt are then introduced in Section 3. In Section 4 it is shown how semistructured data in general and the Xcerpt constructs in particular are visually represented in the language visXcerpt. Section 5 finally gives a summary about further and related work.

2 Positional vs. Navigational Data Selection

Essential to querying semistructured data is the selection of data items in a document (i.e. rooted graph). Most widespread query languages for XML – e.g. XQuery – rely on path selections expressed using XPath (or similar approaches). XPath-like languages provide with constructs like regular expressions and wild cards for specifying paths through a rooted graph. For instance, the XPath expression `/a[b]//c` means "find the document nodes labelled c that can be reached from the document root via a child node labelled a having itself a child node labelled b and having the c-labelled nodes as descendants". Such node selections can be called *navigational*.

For simple queries and transformations, the navigational approach is very natural and results in simple programs. For more complex queries, especially for queries involving several variables, the navigational approach often leads to intricate programs.

Furthermore, the intertwining of construction and query parts in languages such as XQuery and most of its precursors often yields programs that are difficult to visualise properly.

Also, the possibility to specify forward and reverse axes in path languages like XPath might further increase the complexity of query programs and an equivalent query with only forward axes is often more intuitive.

A further important aspect of navigational node selections is that they do not easily support the selection of *several* related nodes at once. Such multiple node selections, however, are rather natural and are required by most non-trivial queries. This is e.g. the case when one looks for bibliography entries combining several aspects such as an author's name, a keyword in the title, and a year of publication. Everyone familiar with bibliographies immediately "visualises" the shape or pattern of such a retrieval request and the respective positions of the variables it refers to. Arguably, pattern-based or *positional* query and transformation languages such as Xcerpt reflect and convey such an intuitive "visualisation".

With the *positional* query and transformation language Xcerpt the nodes to be selected are specified by variables in patterns called *query terms*. Query patterns are related to other patterns called *construct terms* through their common variables. The Xcerpt construct relating a construct term to a query expression consisting of AND and/or OR connected query terms is a *rule*. These concepts are introduced in the next section.

For querying semistructured data, the positional approach has been suggested first with UnQL [4] and XML-QL [11]. In common programming, the positional approach finds its roots in Functional and Logic Programming. Arguably, both query languages QBE and SQL can be seen as positional languages.

3 Xcerpt's Main Constructs

An Xcerpt program may consist of at least one *goal* and of some (maybe zero) *rules*. Goals and rules are built up from database, query and construct terms that are first introduced. Note that besides the "abstract" syntax presented here, Xcerpt also has an XML syntax which is not described here for space reasons.

3.1 Database, Query, and Construct Terms

Common to all terms is that they represent tree-like (or graph-like) structures. Square brackets (i.e. [ ]) denote *ordered term specification* (as in standard XML), i.e. the matching subterms in the database are required to be in the same order as in the query term. Curly braces (i.e. { }) denote *unordered term specification* (as is common in databases), i.e. the matching subterms in the database may be in arbitrary order.

Single (square or curly) braces (i.e. [ ] and { }) are used to denote that a matching term must contain matching subterms for all subterms of a term and may not contain additional subterms (*total term specification*). Double braces (i.e. [[ ]] and {{ }}) are used to denote that the database term may contain additional subterms as long as matching partners for all subterms of the query term are found (*partial term specification*).

Graph structure is expressed using a reference mechanism. The construct `id @ t` is used as a *defining occurrence* of the identifier `id` and the construct `^id` is used as a *referring occurrence*.

Database Terms are used to represent XML documents and the data items of a semistructured database. They are similar to *ground* functional programming expressions and logical atoms. Database terms may only contain the single square and curly braces described above.

A *database* is a (multi-)set of database terms (e.g. the Web). Note, however, that a single database term is often used to represent what is commonly referred to as a "database", as shown in the following example.

Example: a database term representing a bibliography consisting of several books. Note the use of references to share common data. Also note that the author list for a book is ordered while the data in general is unordered:

```
bib {
  authors {
    a1 @ author {
      name { "Serge Abiteboul" }, publications { ^b1, ^b2 } },
    a2 @ author {
      name { "Peter Buneman" }, publications { ^b1 }        },
    a3 @ author {
      name { "Dan Suciu" }, publications { ^b1 }            },
    a4 @ author {
      name { "Richard Hull" },  publications { ^b2 }        },
    a5 @ author {
      name { "Victor Vianu" }, publications { ^b2 }         }
  },
  b1 @ book {
    title { "Data on the Web" },
    authors [ ^a1, ^a2, ^a3 ],
    price { "69.95" }
  },
  b2 @ book {
    title { "Foundations of Databases" },
    editors [ ^a1, ^a4, ^a5 ],
    price { "29.00" }
  }
  ...
}
```

Database terms induce a graph in a straightforward manner. Figure 1 shows a (incomplete) graph representation of the book database of the previous example.

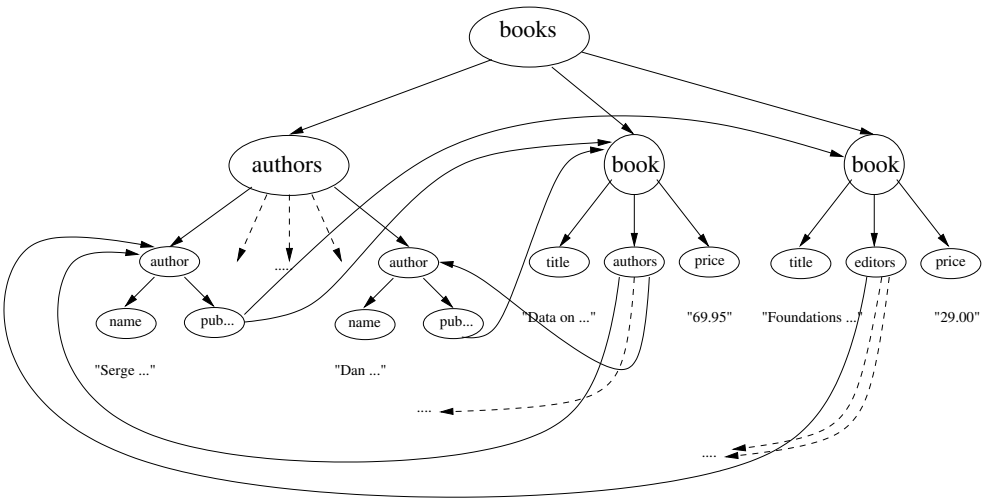

Fig. 1. Graph induced by the book database (incomplete). References to parts not illustrated are shown as dashed arrows.

Note that database terms do not cover all constructs found in XML. Constructs like Attributes or Processing Instructions are intentionally left out because they are either easy to model in the existing database terms or do not add important information to the data represented.

Query Terms are similar to *non-ground* functional programming expressions and logical atoms. Extending the database terms, query terms have the following properties:

- in a query term, *partial specifications* omitting subterms irrelevant to the query are possible (indicated by double square brackets or curly braces),
- in a query term, it is possible to specify subterms at *arbitrary depth* (indicated by the keyword *desc*).
- a query term may contain *term variables* and *label variables* to "select" data from the database (variables are written in upper case letters below)

As Xcerpt queries are pattern-based, a query term should resemble the database as closely as possible, while leaving out such parts that are irrelevant to the query.

The reference mechanism using `^id` and `id @ t` has the same significance as the parent-child edge. In the following example, the right hand side shows a query which matches a parent-child edge with a reference edge in the database.

Example: Left: Select title and author pairs for each book. Right: Select pairs of authors that have written at least one book together.

```
bib {{
  book {{
    var T ⇝ title {{ }},
    authors {{ var A }}
  }}
}}
```

```
bib {{ authors  {{
   author {{
     var Author ⇝ name {{ }},
     publications {{
       book {{
         authors {{ var CoAuthor ⇝ author {{ }} }}
     }} }}
   }}
}} }}
```

The Xcerpt construct $X \rightsquigarrow t$ (read "as") serves to associate a query term to a variable, so as to specify a restriction of its bindings. The Xcerpt construct *desc* (read "descendant" – not illustrated above) is used to specify subterms at arbitrary depth.

Query terms are *unified* with database or construct terms using a non-standard unification called *simulation unification*, which has been investigated in [12]. Simulation unification is based on *graph simulation* [13] which is similar to graph homomorphisms.

The outcome of unifying a query term with a database term are bindings for the variables in the query term. Applying these bindings to the query term results in a ground query term which is simulated (in the sense of [13]) in the database term.

Construct Terms serve to reassemble variable (the bindings of which are specified in query terms) so as to construct new database terms. They may only contain single brackets and variables, but no partial specification or variable restrictions. The rationale of this is to keep variable specifications within query terms, ensuring a strict separation of purposes between query and construct terms.

Example: Create an Author-Title pair wrapped in a "result" element:

```
result {
  var A, var T
}
```

In a construct term, the Xcerpt construct *all* t serves to collect (in the construct term) all instances of t that can be generated by different variable bindings for the variables in t (returned by the associated query terms in which they occur). Likewise, *some* n t serves to collect at most n instances of t that can be generated in the same manner.

Example: Create a list publications for each author and a list of authors for each publication:

```
results {                    results {
  result {                     result {
    var A,                       all var A,
    all var T                    var T
  }                            }
}                            }
```

Example: The following construct term collects all title/author pairs for the previous query:

```
results { all result { var A, var T } }
```

The constructs *all* and *some* n may be nested to form more complex results. The following example shows the usefulness of nesting:

Example: Assuming the previous query, the following construct term collects all titles for each author:

```
results { all result { var A, all var T } }
```

Positioning the nested *all* around the A yields "all authors for each title" as a result:

```
results { all result { all var A, var T } }
```

3.2 Queries

A *query* is a connection of zero or more query terms using the n-ary connectives *and* and *or*. A query is always (implicitly or explicitly) associated with a *resource*. A resource may be the program itself, an external Xcerpt program or an (XML or other) document specified by a URI (uniform resource identifier).

Variables occurring in more than one query terms in an *and* connected query evaluate similar to an equijoin in relational databases.

Example: Query for the prices of books in two different book stores (specified by the resource identifier `A` and `B`).

```
and {
  bib {{
    book {{ var T ⇝ title{{ }}, var Pa ⇝ price{{ }} }}
  }} in http://www.a.com,
  reviews {{
    entry {{ var T ⇝ title{{ }}, var Pb ⇝ price{{ }} }}
  }} in http://www.b.com
}
```

If a query does not explicitly have an associated resource, the resource specification is implicit and inherited from the parent. If none of the parents have a resource specification, or the query does not have a parent, the queried resource is the program itself (i.e. the heads of the rules and possibly database terms contained in the program – see "Rule Chaining" below).

Note that it is possible to use curly and square braces in *and* and *or* connections to specify that the evaluation order is of importance or not. This may serve as an indication to the evaluation engine whether certain optimizations are applicable or not.

3.3 Construct-Query Rules, Goals

An Xcerpt program consists of zero or more *construct-query rules*, one or more *goals* and zero or more database terms. Both rules and goals have the form

$$\text{Construct Term} \leftarrow \text{Query Part}$$

where a construct term is constructed depending on the evaluation of a query part.

Example: A rule that creates a price summary for the books in the two databases A and B:

```
rule {
  cons {
    summary {
      all book { var T, price-at-A { var Pa }, price-at-B { var Pb } }
    }
  },
  query {
    and {
      bib {{
        book {{ var T ⇝ title{{ }}, price{{ var Pa }} }}
      }} in A,
      reviews {{
        entry {{ var T ⇝ title{{ }}, price{{ var Pb }} }}
      }} in B
    }
  }
}
```

A rule can be seen as a "view" specifying how t^c-shaped documents can be obtained by evaluating the query part against a Web resource (e.g. an XML document or a database).

In addition to the form above, goals are always (explicitly or implicitly) associated with an output resource. This resource specifies where to "write" the resulting database terms. If not explicitly specified, the output resource defaults to *stdout*, writing all output to the console.

Rule Chaining. In addition to querying external resources, a query may also be evaluated against the program. In such a case, the heads of the program rules (but not of the goals) are queried and the associated rule is evaluated. Both forward and backward chaining are feasible.

Forward Chaining. In a forward chaining approach, rules are evaluated iteratively against the current set of database terms until saturation is achieved. Forward Chaining is useful for instance for materializing views and for view maintenance.

Backward Chaining. Backward Chaining is a *goal driven* approach. Beginning with the query part of the goal, program rules are selected if they are relevant for "proving" a query term. The query term in question is then replaced by the query part of the selected rule. Backward Chaining is useful when the expected result is small in comparison with the number of possible results of the program.

Backward Chaining in Xcerpt following the SLD resolution used in e.g. Prolog, with some major modifications to cover constructs like *all* and *some* and to cope with multiple results of a simulation unification.

4 visXcerpt: A Visual Rendering of Xcerpt

The main goal of visual languages in general is to ease the use of a technology especially among novice users since it avoids many common errors by abstracting from the textual syntax. The Web context in particular demands for query technology that is easy to use even by non-programmers, since there are always queries not forseen by developers. Hence, a visual language would likely be well accepted among many Web users.

For visual query languages it is considered to be important to have a strong visual relationship between queries and queried data or query results. A natural approach is to provide some sort of example of a valid result as query as first presented in QBE. Xcerpt query patterns with positional variables can be seen as samples of valid source data items, where some parts are left out and others represented by variables. Construction patterns can be seen as samples or templates of result data items.

The syntax and semantics of Xcerpt as a whole is well suited as foundation for a visual language. As a consequence, textual Xcerpt's visual counterpart *visXcerpt* can be conceived as a mere *rendering* instead of a fully novel language. This rendering might be seen as an advanced (because of the dynamic features) layout.

In the following, it is illustrated how the textual Xcerpt constructs have their visual counterparts in visXcerpt. A generic term representation is introduced first, followed by the rule- and query constructs used to form Xcerpt programs and by dynamic aspects of the visual representation.

4.1 Visual Representation of Terms

Xcerpt **terms** (i.e. elements) are visualised as boxes. A term label (or tag) is attached as a tab on the top of its associated box. visXcerpt has features for handling attributes and text. Attributes are placed in a two-column table with names in the left column and values in the right column. The attribute table appears first in a box and is omitted if there are no attributes. Direct subterms (i.e. children) are visualised the same way as sub- or child boxes. Child boxes are arranged vertically in a parent box. For better distinction, they are coloured differently. Figure 2 on the right illustrates this nesting on the Xcerpt term $f[[a\{"TextA"\}, b["TextB"], c[["TextA"]]]]$.

Different box borders are used as visual counterparts to the **Xcerpt parentheses** {{ }}, [[ ]], { }, and [ ]. Ordered or unordered children are indicated graphically by an icon (ascending bars represent ordered, random bars unordered) at the top right corner of a box. Optionally, partial and total matching can also be indicated graphically by an icon (see also Figure 2 on the right).

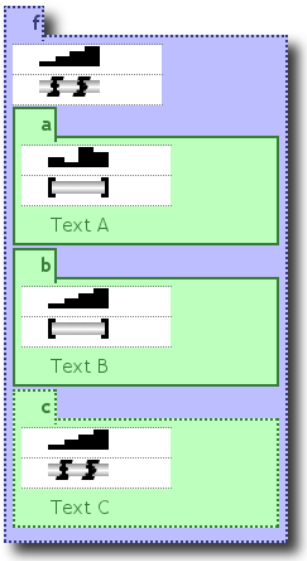

Fig. 2. visXcerpt representation of a term using different combinations of ordered/unordered and partial/total

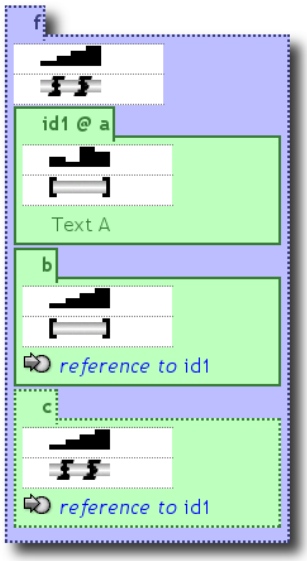

Fig. 3. identifiers appear in the tab, references are visualised by icons

Visual Identifiers and References. Beyond the hierarchical structure that terms can express, Xcerpt provides a reference mechanism based on IDs associated to terms (like $id@t$) and references (like $\uparrow id$).

Figure 3 on the left illustrates the visual representation of a database term containing references and both ordered and unordered content. The references are represented with an icon resembling a pointer and referenced terms carry the anchor name in the title tab.

Note that visXcerpt also provides navigational support for references by representing those constructs as hyperlinks (see *dynamic aspects* below).

Visual Query and Construction Patterns. The constructs presented so far are the foundation of visXcerpt database terms. They can be used to visualise any XML data as well as any Xcerpt data. Visualisation of further Xcerpt constructs are irrelevant for pure data and are distinguished visually

from the former constructs – they use reserved colours (black, white and gray) and in some cases textual adornment with a reserved text style (italic font). Those textual extensions always match to the corresponding Xcerpt keywords.

- *variables* are represented as black boxes with the variable name written in white in the box. If a variable is restricted to a term (by the $\rightsquigarrow$ construct), this term appears within the box of the variable. The variable `A` in Figure 4 illustrates this representation on the example `f[[ desc var A -> a{{ }} ]]`. The variable `A` is restricted to such terms that match with `desc a{{ }}`.
- The *desc* (descendant) construct is rendered as gray box with strong bevelled border (a visual metaphor for depth) and the keyword *descendant* is written in italic at the top (see Figure 4).
- The constructs *all*, *and* and *or* are rendered as white boxes with black border and textual adornment. To further distinguish disjunctions and conjunctions the content of *or* is arranged horizontally while the content of *and* is arranged vertically. A visual representation of *all* and *and* can be seen in Figure 5 (left of the arrow).

4.2 visXcerpt Programs

Visual Construct-Query Rules and Goals. are visualized in visXcerpt by connecting a query part with a construct term by means of an arrow, so as to emphasize the fact that in an Xcerpt rule a result *follows* from a query. As can be seen in Figure 5, the construct term is positioned left of the arrow while the query part is positioned right of the arrow.

Query and/or construct parts may contain resource specifications. Since a resource specification has a scope, it is indicated as a box containing all parts for which they are valid. The resource itself is specified in the title of this box (see Figure 5).

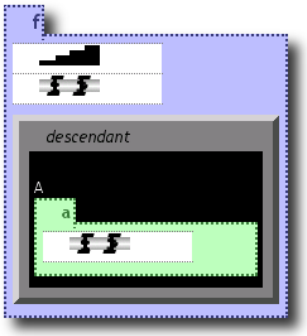

Fig. 4. descendant as bevelled box, variables as black box with white text

Visual Programs are seen as documents of visXcerpt construct-query rules. They are arranged vertically as a list of rules.

4.3 Dynamic Features

A static visualisation as described above is not sufficient for an interactive query system that should provide features which enhance the usability and allow for editing visXcerpt programs. The visXcerpt prototype provides features that allow for easier navigation and improved comprehension (browsing aspects). Furthermore, an editor is provided as well as the possibility to "try out" programs while developing them.

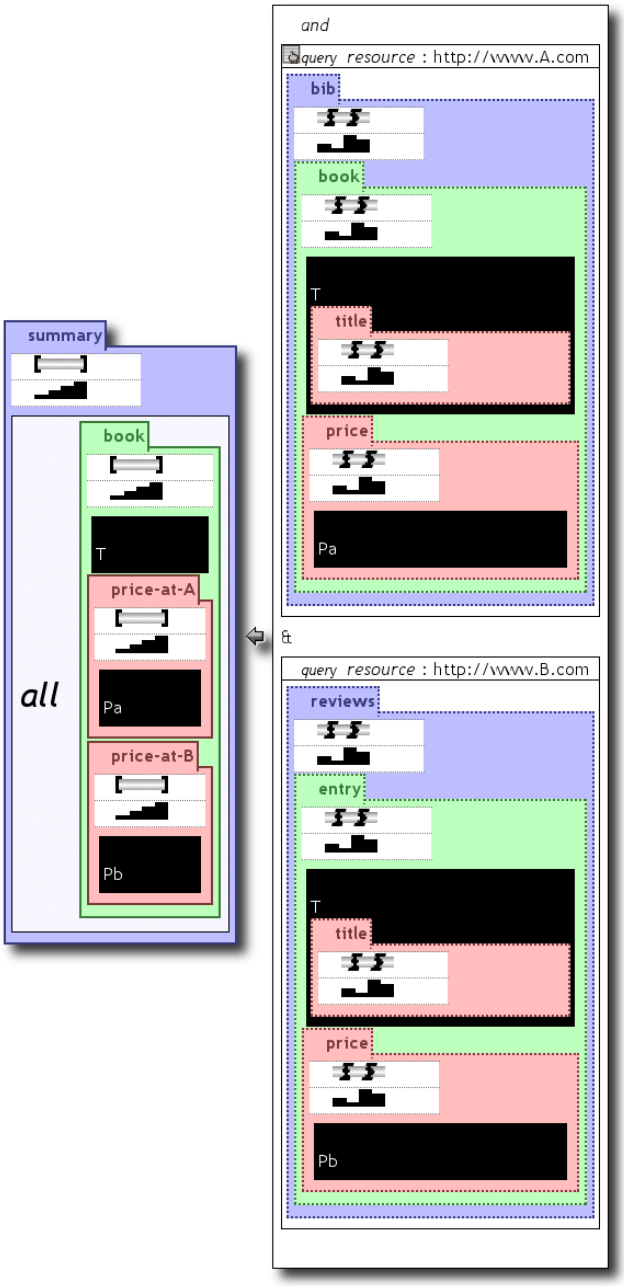

Fig. 5. A visual representation of the Xcerpt rule from Section 3.3. The construct term is left of the arrow while the query part – consisting of an *and* connection of two query terms – is right of the arrow. Note that the Xcerpt-specific constructs not found in database terms (e.g. variables) are always displayed in shades of black and white.

Browsing Aspects. When viewing (and editing) documents, an important aspect are properties that allow navigation and different views upon the data. In visXcerpt, such properties are referred to as *browsing aspects*. In particular, visXcerpt provides means for:

Partial Viewing. For large documents, only a part is displayed in the viewer. Vertical and horizontal scrollbars allow to move the current view.

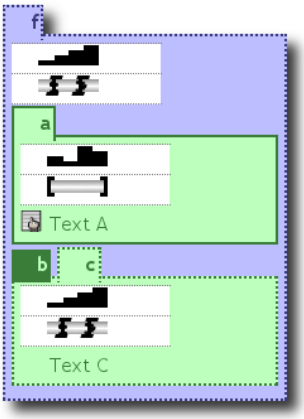

Fig. 6. information hiding: element "b" is folded

Information Hiding. In large documents, it is desirable to be able to hide such information that is irrelevant for the current task. In visXcerpt, clicking on the title tab of an element "folds" the element together with all its contents such that only the title tab remains (in a shaded color). Any subsequent element boxes below the same parent element move upwards such that their title tab is besides the hidden elements (Illustrated in Figure 6 on the right).

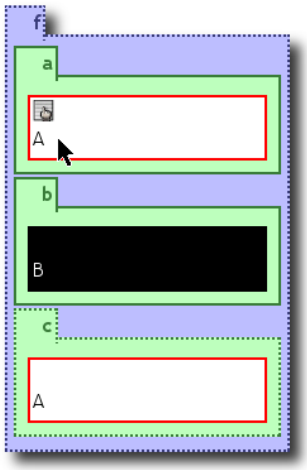

Fig. 7. variable highlighting of the variable "A"

References. Visually depicting references with icons and identifiers only (as described above) is dissatisfactory, since it does not model the graph structure appropriately. Instead of further depicting references visually, visXcerpt "moves them into hyperspace" by representing them as hyperlinks. That is, by clicking on a reference the visualisation scrolls or focuses on the occurence of the referenced element. Hovering with the pointer above a term with ID highlights all occurrences of references to it. Backward navigation to references is supported through a popup menu of elements containing an ID.

Variable Highlighting. In (vis)Xcerpt rules, all variables that appear in the head of a rule are also required to appear in its body. Moreover, the same variable may occur in several parts of the body, even several times within the same query term. To support the user in designing visXcerpt rules, all occurrences of a variable are highlighted by inverting its color when the mouse hovers over one occurrence of the variable (see Figure 7 on the left). This eases the comprehension of term equality in positional queries and thus allows the user to recognize connections between different parts of a rule or within a term.

The reference visualisation and variable handling is similar indeed and future implementations of variable visualisation may rely on the more general reference mechanism.

Editing Capabilities. As visXcerpt is an editor for tree structured data, many of the editing capabilities commonly found in plain text editors have only limited applicability. Thus, in addition to common text editing primitives (like "cut", "paste"), the visXcerpt editor provides primitives that are suitable for insertion of subtrees and nodes (e.g. "paste into at beginning/end" or "paste before/after", illustrated in Figure 8 on the right).

When build documents, the visXcerpt editor follows a "copy and paste" paradigm with a template area where templates of common elements like "element" or "variable" can be copied from and inserted into the edited document. A "drag and drop" paradigm has also been considered but is not yet implemented due to technical reasons.

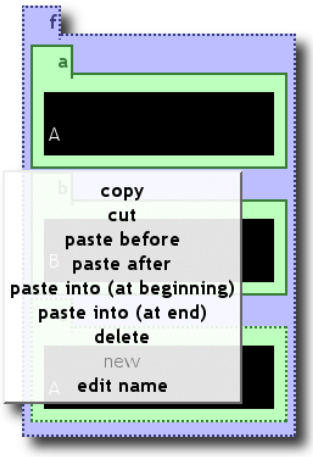

Fig. 8. Editing capabilities: context menu

Interactive Queries. When designing programs, it is often necessary to be able to run the program for test purposes as well as restrict the test to certain parts of the program. The visXcerpt interface allows both:

- A program can be evaluated at any time, provided its semantics is meaningful (the visual interface ensures that the syntax is *always* correct). For a program to be evaluated, it must contain at least one goal. Depending on the output resource specified for the goal, the result of the evaluation is either displayed in a new windows in the generic visXcerpt term representation or written to the specified output resource.
- A single query term can be evaluated against the program to test only specific parts. This query term is evaluated against all rules and the result is displayed as a disjunction of alternative variable bindings in a new window.

5 Related and Future Work

There is a large number of XML query languages available on the Web, most of them based on a navigational selection of nodes. Most notably, the W3C has issued the XQuery, XSLT and XPath recommendations [14]. The pattern-based approach to querying semistructured data has first been presented in the language UnQL [4]. However, UnQL rules may not be connected by chaining and

lack many of the constructs found in Xcerpt. Several publications concerning the language design and semantics of Xcerpt are available, in particular [5,15,12,16].

Visual XML query languages that the authors are aware of are XML-GL [6], GraphLog [7], VXT [8], BBQ [9] and Xing [10]. Most of these languages visualize the XML document as a tree (i.e. nodes connected with arrows or similar). While on first look this appears to be very concise, it does not scale well to larger documents and queries. Thus, visXcerpt uses the concept of nested boxes as visual representation, which is borrowed from the language Xing and enhanced in many ways in the visXcerpt viewer/editor.

Work on Xcerpt is currently conducted to formally provide a model-theoretic semantics (see [16]) and a reasoning calculus. Future work will investigate Xcerpt as a reasoning language in a Web environment and add additional language features like arithmetics, basic and complex types, constraints, etc.

The visXcerpt prototype will be extended by adding improved browsing facilities, like browsing from an element to such elements that refer to it. Furthermore, investigating suitable commands for editing tree- and graph-structured data is of major interest. As the current visXcerpt editor is implemented prototypically in HTML and JavaScript, a more efficient implementation is also sought for, possibly by extending already-existing XML editors.

References

1. W3 Consortium http://www.w3.org/TR/REC-xml: Extensible Markup Language (XML) 1.0, Second Edition. (2000)
2. W3 Consortium http://www.w3.org/TR/owl-ref/: OWL Web Ontology Language Reference. (2003) W3C Working Draft, 31 March 2003.
3. Bry, F., Kraus, M.: Position Paper: Style Sheets for Context Adaptation. In: W3C Delivery Context Workshop. (2002)
4. Buneman, P., Fernandez, M., Suciu, D.: UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. VLDB Journal **9** (2000)
5. Bry, F., Schaffert, S.: A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In: Proc. Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web. (2002) (invited article).
6. Ceri, S., Damiani, E., Fraternali, P., Paraboschi, S., Tanca, L.: XML-GL: A Graphical Language for Querying and Restructuring XML Documents. In: Sistemi Evoluti per Basi di Dati. (1999)
7. Consens, M., Mendelzon, A.: Expressing Structural Hypertext Queries in GraphLog. In: Second ACM Hypertext Conf. (1989) 269–292
8. Pietriga, E., Quint, V., Vion-Dury, J.Y.: VXT: A Visual Approach to XML Transformations. In: ACM Symp. on Document Engineering. (2001)
9. Munroe, K.D., Papakonstantinou, Y.: BBQ: A Visual Interface for Integrated Browsing and Querying of XML. In: VDB. (2000)
10. Erwig, M.: A Visual Language for XML. In: IEEE Symp. on Visual Languages. (2000) 47–54
11. Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Suciu, D.: A Query Language for XML. In: Proc. of Eighth Int. WWW Conf. (1999)

12. Bry, F., Schaffert, S.: Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In: Proc. Int. Conf. on Logic Programming (ICLP). LNCS 2401, Springer-Verlag (2002)
13. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web. From Relations to Semistructured Data and XML. Morgan Kaufmann (2000)
14. World Wide Web Consortium (W3C) http://www.w3.org/. (2002)
15. Bry, F., Schaffert, S.: The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In: Proc. 2nd Int. Workshop "Web and Databases". LNCS 2593, Erfurt, Germany, Springer-Verlag (2002)
16. Bry, F., Schaffert, S.: An Entailment for Reasoning on the Web. Technical Report PMS-FB-2003-5, Institute for Computer Science, University of Munich (2003)